Volume

1

New User To Technical Expert – Solaris Bookshelf

Stewart Watkiss

Solaris User Guide

Copyright

Stewart Watkiss

Contents

Preface to Bookshelf ...... **vii**

Introduction to this book ...... **ix**

Chapter 1 : **THE UNIX OPERATING SYSTEM** **1**

What is an Operating System **3**

More About UNIX **4**

The difference with Solaris **5**

Chapter 2 : **GETTING STARTED** **7**

The Login **9**
Logging Directly into a Local Machine 9
Logging in Over a Network 11

Exiting (logout) **13**

Important Points for New Users **14**

Chapter 3 : **SHELLS** **15**

What is a shell? **17**
The Various Different Shells 17
The Shell Prompt 19

Chapter 4 : **COMMANDS AND PROGRAMS** **21**

What are Command and Programs? **23**

Command Basics **23**
Format of Commands 23

Help with Commands **24**
Finding Files and Commands (find) 28
A few useful commands 28
Clear 29
Echo 29

Chapter 5 : **FILES AND DIRECTORIES** **31**

What are Files and Directories? **33**

Relative Directories....35
Special Directories....35
Moving about the directories (cd)....35
Listing the contents of the directories (ls)....36
Referring to files within a directory....38
Hidden Files....38
Lost + Found....38

Chapter 6 : **WORKING WITH FILES....41**
Making a new directory (mkdir)....43
Making a new file (touch)....43
Removing a directory (rmdir / rm)....43
Removing a file (rm)....44
Moving / Renaming a file or directory (mv)....44
Copying a file (cp)....44
Viewing the contents of a text file (cat)....45
Viewing the beginning / end of a text file (head / tail)....45
Checking the type of file (file)....46
Printing a file....46

File Permissions....46
Changing File Permissions (chmod)....47
Changing the file owner (chown)....49

Chapter 7 : **MAKING THE MOST OF UNIX COMMANDS....51**

Using command switches....53
The pipe command (|)....53
Redirecting stdout, stdin and stderr (> <)....54

Chapter 8 : **MANAGING YOUR ACCOUNT....57**

Normal Vs' Root Username....59

Shell Variables....59

.profile....62

.kshrc....64

History....65

Chapter 9 : **THE VI TEXT EDITOR....67**

Editor Modes....69

Worked Example with VI....70
Starting VI....70
Inserting a New Line....70

Moving Around the File 71
Deleting Letters and Inserting a New Word 71
Save the Document 71
Replacing a Letter 71
Search and Replace 72
Save and Exit 72

Preferences 72

Glossary 75

Command Summary 81

Index 87

Preface to Bookshelf

This is the second UNIX book that I’ve written. The first is a book on AIX and incorporates the full range of New User to Technical Expert in a single book. The book started out as a aide-mémoire as part of my revision towards becoming an AIX Certified Advanced Technical Expert. I passed the exams and continued writing until it took the form of a complete book on the subject.

After completing my AIX Certification and Book I set myself a second challenge of applying my acquired UNIX skills to learn Solaris. This bookshelf is intended to pass that on to others that may be interested. Rather than writing this as a single book I decided for the bookshelf approach, as it is easier to manage as several smaller books.

This was written whilst I had hands on role implementing some new services on Solaris. This has given me practical experience of how the operating system can be used and in some areas how it really works as opposed to what the manuals say.

Introduction to this book

This book is written for a new user to UNIX. The target audience of this book would typically be someone with little or no experience in UNIX, but maybe a bit of background using other operating systems (e.g. Windows 95/98).

All sections are explained in full and therefore no previous experience of any other environments are necessary. The book also includes information on some of the many commands available. It may therefore be useful for someone with previous experience of UNIX.

Most of the commands used in this section are the same as used on other UNIX like operating systems such as AIX and Linux.

There generally appear to be too different approaches when creating a user guide for a UNIX based computer. One is to explain how to use a graphic interface, explaining what drag and drop means and how to use a mouse, the other explains the commands that allow a user to learn how to operate using a command line. This book is based on the latter; although I do make reference to X-Windows and CDE I am assuming that you do know what a mouse is and how to click on pull down menus.
The reason I have gone down this path is that I see the novice Solaris user as being someone that will eventually become a system administrator and not someone that will be just clicking mouse buttons all day long. Whilst Solaris may provide much of the functionality that is required for running GUI based programs I think that Linux is a more natural choice for the complete beginner who does not have any desire to learn the power of the command line. That is not to say Linux is not suitable for those who prefer the command line, but that it is much more popular as a graphical desktop than Solaris. Although Solaris does win back users with it’s support for high end CAD and Graphics Design applications.

Often when people write a separate user guide and system administrators guide they write in an overlap so that you don’t have to have both books to understand more about the interaction between a System Administrator and a User. I have not. Providing the books for free means that anyone that has a copy of the User Guide can equally obtain a copy of the System Administrators guide and vice-versa. So if something is not explained in the User Guide then pick up Volume 2 and you’ll probably find it in there.

Chapter 1

The UNIX Operating System

This chapter explains what UNIX is and why you need to know about it.

What is an Operating System

UNIX is an operating system, that is a layer of software that provides a path for applications to control and communicate with the hardware.

Hardware refers to the bit's you can actually see and touch - Disk drives, Monitors, Keyboards, CPU etc.

Software refers to programs or pieces of code that make the computer perform a required function. In fact software is what differentiates a computer from other electronic devices. A computer has the ability through using different software to perform different functions. Compare this to an electronic typewriter that is only capable of doing what the hardware is coded to do (type letters directly onto paper). Nowadays it is harder to distinguish the difference as Word Processors can perform the function of a typewriter but can have the ability to do other things, such as transfer the files to a computer or save to disk.

Operating Systems are what tie the Software Applications and Hardware together.

To explain this a little clearer I'll use an analogy to a taxi. The taxi itself would be the hardware part of the computer, the people travelling in the taxi would be the applications and the taxi driver would be the operating system.
Imagine then that you are requested by your boss to attend a meeting at a different location and that you should take a taxi. This would be like a user requesting that an application performs a certain function. You would then request a taxi and get into it. You would not be allowed to drive the taxi yourself so would make a request to the driver (the operating system) to take you to a certain destination. The driver would drive the taxi (control the hardware by using the pedals, steering wheel etc.) to the destination requested by you. Once you reached your destination you would leave the taxi allowing it to transport other people.

You can see that this would shield you from having to know how to drive the taxi, you would not need to know whether it was a diesel taxi or a petrol taxi, or whether it was a manual or automatic. All you need to be able to do is to talk with the driver and explain where you wanted to go.

In this way if an application needs to write to a disk, it does not need to know the type and size of the disk, it does not need to know how to move the head backwards and forwards. All it needs to do it to send a request to the operating system asking it to write the required data to disk and the operating system will worry about all the details.
With multitasking operating systems such as UNIX, several programs can be running at once. The operating system is also required to control the applications so that they don't interfere with each other. It does this by controlling which applications should be running in the processor, which memory has been allocated to it and when it needs to step back and let another program run in the processor.

The following list are some of the operating systems that you may have heard of. UNIX; MS-DOS; Windows 3.1 / 98 / NT and Apple Macintosh.

It is worth noting that whilst many of the operating systems come with bundled applications for example word processors, basic graphics editors / viewers, calculator etc. the applications are not actually part of the operating systems. If these applications were removed the operating system would still function properly and other applications could still communicate with the hardware via the operating system.

More About UNIX

UNIX was originally developed during the early 1970's at AT&T's Bell Laboratories. The main developers were Ken Thomas and Dennis Ritchie.

UNIX was developed as one of the earliest multi-user multitasking (time-sharing) operating systems. Earlier computers were only able to handle a single user running a single program at any one time. These new generation of computers had to be able to run more than one program by more than one user. This was done using time-sharing where each program (more accurately a thread) would run in the processor for a certain length of time, before being removed from the processor to let another program run. When done fast enough this gave the illusion that several programs were running simultaneously. With multiple processors it's possible to run a true multitasking environment where multiple threads can be running at simultaneously in the separate processors.

Where UNIX was different from other operating systems at the time was that it was designed as an open standard. Compared with proprietary operating systems, which could only run on a certain computer, UNIX could be modified to run on a variety of different computers from different manufacturers. This no longer tied the operating system to any particular manufacturer of computer and was ultimately one of the reasons for its success.
The operating system became an open free operating system that was developed further by Universities and companies. The most notable of these was Berkeley University, in fact many of the features in UNIX are referred to as to BSD standard, referring to Berkeley Software Distribution.

Unfortunately this open standard also allowed companies to go there own way and develop different "improvements" to the operating system. This in turn brought about incompatibilities between the different versions. A list of some of the better known variants of the UNIX operating systems are listed:

Company / Developer	UNIX Operating System
Sun	SunOS / Solaris
IBM	AIX
Hewlett-Packard	HP-UX
Linus Torvalds	Linux

Note that there are two operating systems next to Sun (SunOS and Solaris). As far as you are concerned as either a UNIX user or a system administrator these are just different names for the same operating system.

The difference with Solaris

Solaris is Sun's version of UNIX designed primarily to run on Sun's own SPARC based workstations and servers. It is now also available to run on x86 based computers as well.

Solaris is a popular variant of UNIX and is well supported by most software writers. It is the most popular commercial UNIX operating system. Solaris does fall down when compared with some variants, as it does not have some of the System Administration tools available in other UNIX's (such as AIX's SMIT). Where it falls down in other areas it is normally possible to get a third party alternative, although this does add additional cost.

There are also a lot of common things introduced to make some different UNIX Operating Systems look and feel the same despite being from different companies. One of these is the Common Desktop Environment (CDE). This is an X Windows, Window Manager developed jointly between IBM, HP, Sun and SCO designed to give a common theme and common set of basic applications across different platforms.

Chapter 2

Getting Started

This chapter provides the initial guidance, for when a new user first tries using Solaris.

The Login

This section includes some of the basic information a new user needs when first logging in and getting around in a UNIX environment.
This assumes that you have a pre-installed system and that you have been given a username and password to login to the system. If you are going to have sole responsibility for the machine then you may need to look at the System Administrators Guide for details on installing the operating system.

UNIX machines are often used by lots of different people. This differs from PC's which are often used solely by one person. To ensure that only those that are authorised to use the computer, and to ensure that files can only be read by people who they are intended to it is necessary to "login" to the computer. The most common method for logging into a computer is to give a username and password. It is not possible to issue any commands or access any files until a valid username and password are entered. This is different to Windows 9x where the cancel button can be pressed but still give you access to the computer.

It is important that the password is kept secret as this is the basis of all security regarding your identity and files. Further details on the importance of usernames and passwords are provided in the Security section of this book.

There are a number of ways of logging in to a UNIX system. The two most common methods are mentioned, logging in directly onto a local machine or by connecting over a network using telnet.

Logging Directly into a Local Machine

When logging into a local machine, you will have a screen and keyboard attached directly to the UNIX computer. The Computer will normally be either text based or a Graphical Workstation. The login prompt should be already on the screen. This will prompt initially for your username. You will have been provided with the username from your system administrator. Usernames and passwords are both case sensitive and therefore "stewart" is not the same as "Stewart" (this will typically be all in lower case i.e. "stewart").

```



Banner Message
login:
```

Type in your username
For a graphics login then hit the "TAB" key to move to the password field. For text logins hit the "ENTER" key to be prompted for your password.
You should then enter your password and hit "Enter". Passwords will not show on the screen and are case sensitive.
You may be required to change the password on your first use.

When logging onto a UNIX system care should be taken when entering the password as it does not display and is very difficult to correct if you make a mistake. Sometimes you may see an ‘X’ on the screen for every key entered but not always. If you do make a mistake you could try pressing the “DELETE” key but this does not always produce the desired effect.

Depending upon the security settings of the machine you are trying to log onto you may only a few attempts at entering the correct password. More than the maximum attempts and your password will be revoked preventing you from logging in in future. If this happens you should contact the system administrator to get your password and login count reset.

If you enter the username or password wrong little information will be given to the actual cause of the error. For example it may say “You entered an invalid login name or password” and prompt you to retry. This is deliberate as it makes it harder for any unauthorised people to try and login by guessing passwords, however this also makes it harder to see what you entered incorrectly. If this does happen then re-enter your username and password slowly and carefully to ensure that you are not hitting the wrong key.

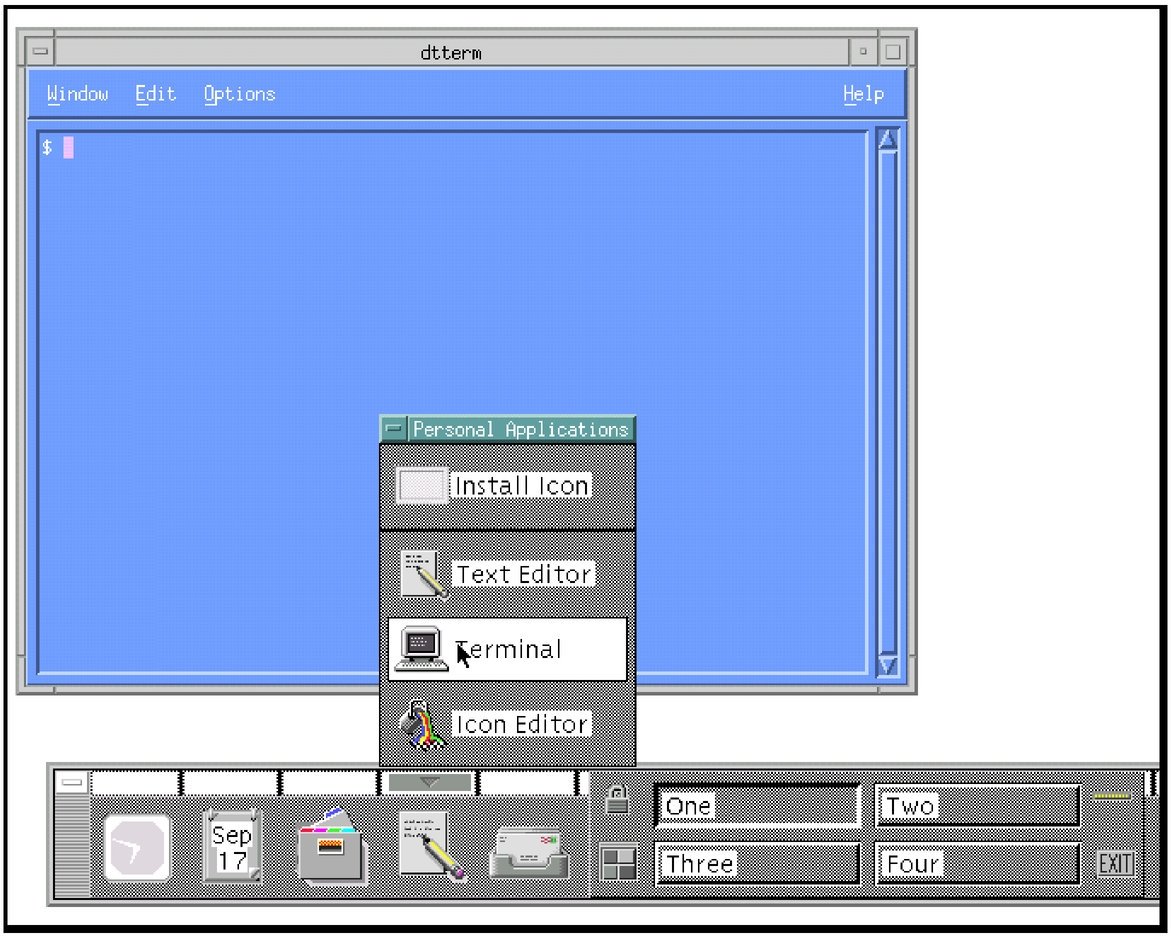
```
Banner Message
login: invaliduser
Password:
Login incorrect

Banner Message
login:
```

You should then get either a text screen for a text based screen or a Graphical Windows system if the system is set to use a graphical login.
If you are at a text screen then you are ready to proceed to the next section. If you have a graphical screen then we need to have an active "Command Window" to continue. If you already have a text window within the graphical display then this will probably be what is required if not then you need to start a "Command Window".

```
Banner Message
login: stewart
Password:
Last login: Mon Mar 26 10:56:10 from 192.168.2.3
Sun Microsystems Inc.   SunOS 5.6       Generic August 1997
$
```

The command window can be accessed by either right clicking the mouse button on the main part of the screen and selecting one of: Terminal; Command; Prompt; Term etc. or if you are running CDE it’s one of the ICONS available in the same group as the text editor. The Command Window should have a prompt on the screen this may be the ‘$’ sign as shown above or a ‘>’ or anything that the system administrator has set for you.

Once you have a text “Command Window” you can proceed to the next section.



The above screen shot shows where the terminal icon is on the CDE menu. It also shows the terminal running in the background.

Logging in Over a Network

There are several ways of connecting to a UNIX computer over a network. I am assuming the TCP/IP protocol is running on the UNIX machine and that TCP/IP is running on a PC or similar from what you will be logging in from.

Depending upon the type of computer that you are connecting from it is possible to do different things. The standard available across all platforms is the ability to get a text command screen that allows commands to be run and text output to be seen. It is possible if using another UNIX type environment to redirect graphics applications from the remote machine to the computer that you’re connecting from. With Windows and OS/2 this can also be achieved by using emulation software such as Exceed, which is developed by Hummingbird.

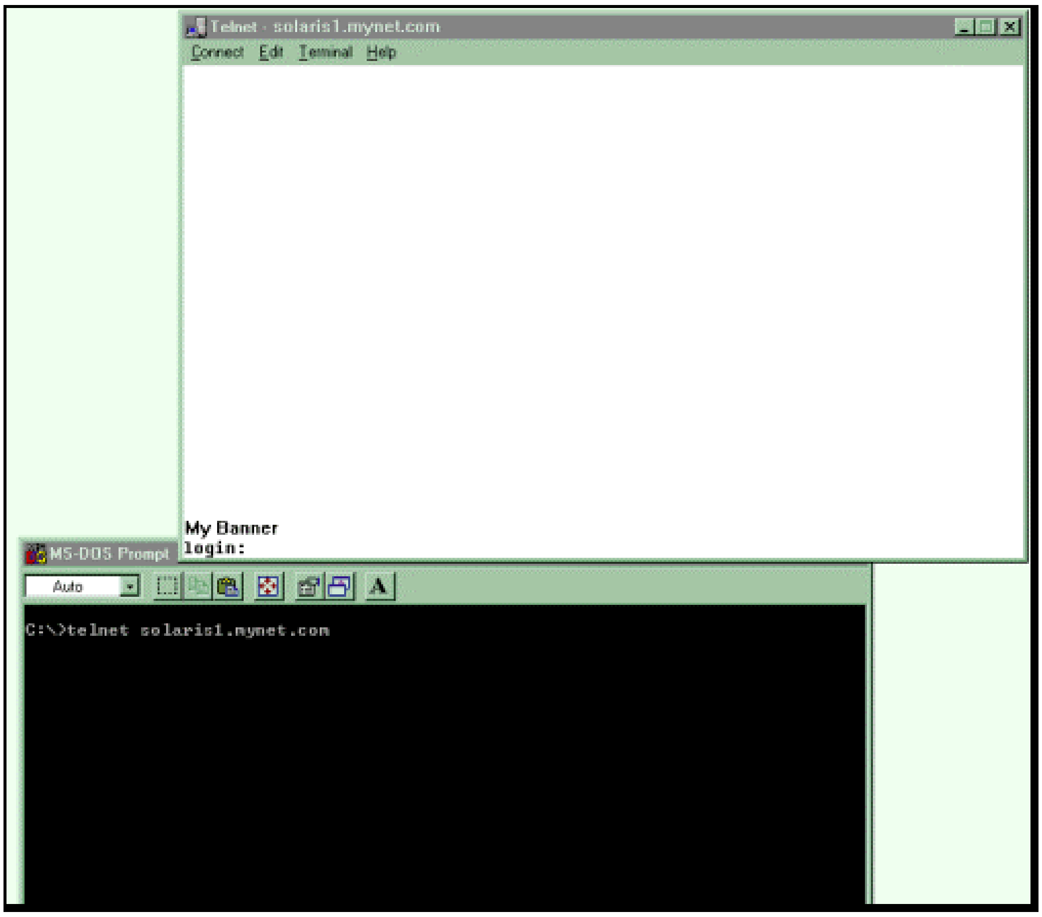
Whilst there are a few ways of connecting (e.g. rlogin, rsh, rexec, telnet) I will be using telnet, which is probably the most widely available of all the methods.

Telnet is provided as standard on the following platforms: All types of UNIX, Linux, Windows 95 and later, OS/2 and even some "dumb terminals".

Telnet is started by issuing the command

```
telnet machine
```

from a command prompt (MS-DOS prompt for Windows 95/98). Where machine is either the IP address or the hostname for the computer.



```
My Banner
login:
```

```
C:\>telnet solaris1.mynet.com
```

e.g. IP address 10.12.142.7 might have a hostname of solaris1.mynet.com . The hostname or address will be provided by the system administrator. The hostname will generally be the computer name followed by the network domain that it is connected in. See the Networking section for a further explanation of the TCP/IP protocol.

You should then get a login screen to the UNIX machine. This should show the name of the computer followed by a login prompt requesting a username.
You will have been provided with your username, this is case sensitive and therefore "stewart" is not the same as "Stewart" (this will typically be all in lower case i.e. "stewart").

Type in your username

Then hit the "ENTER" key to be prompted for your password.
You should then enter your password and hit "Enter". Passwords will not show on the screen and are case sensitive.
You may be required to change the password.

When logging onto a UNIX system care should be taken when entering the password as it does not display and is very difficult to correct if you make a mistake. Sometimes you may see an 'X' on the screen for every key entered but not always. If you do make a mistake you could try pressing the "DELETE" key but this does not always produce the desired effect.

Depending upon the security settings of the machine you are trying to log onto you may only a few attempts at entering the correct password. More than the maximum attempts and your password will be revoked preventing you from logging in in future. If this happens you should contact the system administrator to get your password and login count reset.

If you enter the username or password wrong little information will be given to the actual cause of the error. For example it may just say "Authorisation Failed" and prompt you to retry. This is deliberate as it makes it harder for any unauthorised people to try and login by guessing passwords, however this also makes it harder to see what you entered incorrectly. If this does happen then re-enter your username and password slowly and carefully to ensure that you enter them correctly.

```



Banner Message
login:
```

After entering your username and password correctly you will end up at a command prompt.

```


Banner Message
login: stewart
Password:
Last login: Mon Mar 26 10:56:10 from 192.168.2.3
Sun Microsystems Inc.   SunOS 5.6       Generic August 1997
$
```

Exiting (logout)

Just as it is required to login, it is important that after you have finished you should logout of the computer. This prevents anyone else from coming up to your terminal after you have left pretending to the computer that it is you that is telling it what to do.

This is done by entering
`logout`

from the command line. For a user at a graphical screen it may be necessary to click on the icon marked "EXIT" or one marked with an ' X'.

Another way of logging out of a logged in session / or of cancelling an application requiring further input is to use CTRL-D.

If you are logged in remotely, or are using a virtual terminal then you should type:

```
exit
```

to close the current session.

Important Points for New Users

There are a few oddities of the UNIX operating system that may catch a new user unawares.

Probably the most prevalent of these is that UNIX is case sensitive. With some other operating systems entering “telnet” would have the same effect as if you entered “TELNET” or indeed “TeLnEt”. This is not the case with UNIX where almost everything is case sensitive. It is most common for directories and files to be all in lower case. Unless you have a good reason to do it differently I’d suggest that it is a good idea to use lower case throughout.

A second difference is that the UNIX directory path separator is the ‘/’ (forward slash) as opposed to DOS, Windows and OS/2 which all use the ‘\’ (backward slash). This can be a constant point of frustration for someone that spends a lot of a time switching between UNIX and other operating systems in remembering to use the correct command directory path separator. The UNIX file structure has a number of advantages, which are covered later.

Depending upon your method of connecting to the computer some of the keys on the keyboard may not work as expected (or indeed at all). Some of the keys to be aware of are “BACKSPACE”, “DELETE”, and the arrow keys. If the “BACKSPACE” key doesn’t work then try the “DELETE” key and visa-versa.
It can be very frustrating if the backspace key doesn’t work. Fortunately this is something that can be easily changed. To set the backspace key to delete then enter the following command

```
stty erase <BACKSPACE>
```

The <BACKSPACE> part of the command is where you should press backspace to set the key. The backspace key will normally display as ^h.

The arrow keys are not really needed on a UNIX system as most programs are written to accept normal keys as cursor keys.
These problems can sometimes be resolved by changing the terminal type or by re-mapping of the keyboard, which are described later.

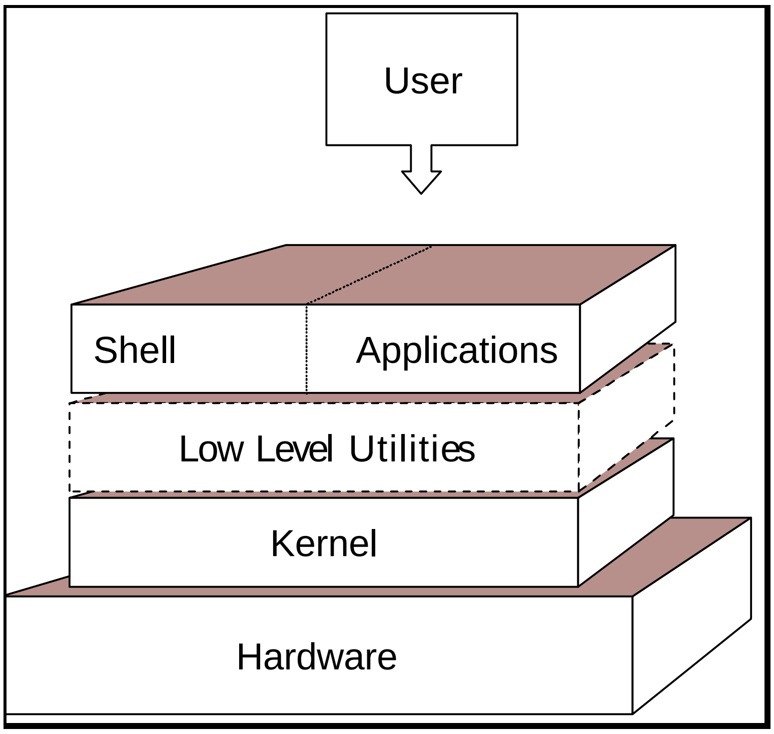
Chapter 3

Shells

The shell is the basic interface between the user and the operating system. This chapter will explain what it is and how to get the most out of the shell.

What is a shell?

If you've followed the instructions above you'll be in a UNIX shell now. Whilst this is what most people see as being the UNIX operating system it is in fact a program that is running on top of the operating system. To take a basic view of how UNIX is built up see the diagram below:



The kernel is the heart of the operating system. This is a process that runs continuously managing the computer.
The kernel is a very specific task so to allow programs to communicate with it there are a number of low level utilities that provide an interface between the application and the kernel.
The shell is an application that allows users to communicate with the computer. It is a text based application that allows programs to be started and tasks to be run.

This can also be visualised by likening the software to a nut (the type that does grow on trees). The kernel is the inter most workings, with the shell on the outside protecting the kernel against the user.

The Various Different Shells

In the same way that different variants of UNIX were developed there are also different variants of the shell.

Here's a list of the most common UNIX shells:

Name of shell	**Command name**	**Description**	**Installed by default on Solaris**
Bourne Shell	sh	The most basic shell available on all UNIX systems	☑ Default Shell
Korn Shell	ksh	Based on the Bourne shell with enhancements	☑
C Shell	csh	Similar to the C programming language in syntax	☑
Bash Shell	bash	Bourne Again Shell combines the advantages of the Korn Shell and the C Shell	☒
tcsh	tcsh	Similar to the C Shell	☒

On a standard setup you will normally have the Bourne shell by default. You can switch to any other installed shell by entering the command name (e.g. entering csh will give the C Shell). The users default shell for a user is contained within the /etc/passwd or can be changed by the system administrator.

The shell is more than just a way of typing commands. It can be used to stop, start, suspend programs and by writing script files it becomes a programming language in itself.

More details of the shells are listed below.

Bourne Shell - This is the oldest shell and as such is not as feature rich as many of the other shells. It's feature set is sufficient for most programming needs however it does not have some of the user conveniences that are liked on the command line. There is no option to reedit previous commands or to control background jobs. As the bourne shell is available on all UNIX systems it is often used for programming script files as it offers maximum portability between older UNIX machines. This is the default shell used by Solaris when newly installed.

Korn Shell - This is based on the Bourne shell. One enhancement that is particularly useful is it’s command-line editing facility. It is possible using either vi or emacs keys to recall and edit previous commands. There are also more powerful programming constructs than the bourne shell, however these are not as portable to older machines. To run the Korn shell you can run ksh from the normal shell. Unless specified I will assume the Korn shell is being used for the rest of this book.

C Shell - The c shell syntax is taken from the C programming language. As such it is a useful tool for anyone familiar with programming C.

Bash Shell - The Bash shell is a combination of features from the Bourne Shell and the C Shell. It's name comes from the Bourne Again SHell. It has a command-line editor that allows the use of the cursor keys in a more "user friendly" manner than the Korn shell. It also has a useful help facility allowing you to get a list of commands by typing the first few letters followed by the "TAB" key.

tcsh - This is a different shell that emulates the C Shell. It has a number of enhancements and further features even than the bash shell.

The Shell Prompt

When logged into the shell you will normal see one of the following prompts: $, % or #.
This is an indication that the shell is waiting for an input from the user. The prompts can be customised but generally the last character should be left as the default prompt character as it helps to distinguish between what shell you are running and whether or not you are logged in as root.

The Bourne, Korn, and Bash shells all accept a similar syntax and unless you are using one of the advanced features you do not necessarily need to know which one of them you are in. If however you are in the C or tcsh shells this uses a completely different syntax and can require commands to be entered differently. To make it a little easier these have two different prompts depending upon the shell.

The default prompts are:

$	- Bourne, Korn and Bash Shells
%	- C Shell

When logged into the computer as root, the user should take great care over the commands that are entered (further information about the root user is included in the System Administrator book). If you enter something incorrectly you could end up damaging the UNIX installation files or even delete all the data from a disk. For this reason the prompt is different when logged in as a root user as a constant reminder of the risks.

The default prompt for root is the hash sign # this is regardless of the shell being used.

Chapter 4

Commands and Programs

This chapter explains what a command or program is. It explains how to run a command and how different commands can be used to interact to provide extra functionality. It also tells you where to go for help if you get stuck.

What are Command and Programs?

Ever time you do something on a computer you need a program to run to perform the task. A command is a program and a lot of times the names can be used interchangeably, however the names are often used in different circumstances.

Normally if you run a program on the command line that runs, (optionally) provides an output and then exists you would refer to this as a command. However if you ran a program that needed user input or that ran as a graphical program under X-Windows then this would be referred to as an application or as a program.

Command Basics

UNIX commands are not necessarily the easiest commands to remember. They are designed to be short commands to reduce the amount of typing:

e.g.
- ls - List directory contents
- cd - Change Directory
- cp - Copy
- pg - Show output one Page at a time

This can make it a little hard to remember but does save on typing when entering a large number of commands. The commands are also highly customisable by having a large number of options that can be entered on the command line however many other functions are provided through a pipe. For example most programs will output to the screen without worrying about how many screen-fulls there are. However all that is needed so that you can view more than one page is to pipe the output through the pg command. This is explained later.

Format of Commands

Generally most UNIX commands are designed to be run from a command line and accept a number of options or arguments when run. This allows commands to run without interaction from the user (often required on tasks designed to run in the background). These are usually in the format:

command option(s) argument(s)

an example of this would be the ls command. The ls command will be explained later, however for now it is sufficient to know that the ls command will list the contents of a directory and will accept certain options and arguments. One option is the -l which means provide more details about the files and another is -a which shows all files even hidden ones. The argument provided is a file or directory name.

```
ls -l /home/stewart
```

will show the contents of my home directory. The -l is an option in that it changes to way the program runs and the argument is /home/stewart which tells the program what directory to look in. The ls command doesn't require any options or arguments. For example the following are also perfectly valid commands.

`ls /home/stewart`	shows a brief listing of the specified directory
`ls -l`	shows a full listing of the current directory
`ls`	shows a brief listing of the current directory

Also if more than one option is required there are two ways of specifying them. Either individually as separate options i.e..

```
ls -l -a /home/stewart
```

or combined i.e.

```
ls -la /home/stewart
```

both the above will run identically.

Help with Commands

The universal help tool in UNIX is the Manual Pages. These are accessed by using the “man” command. If you know the command that you want help on then enter man followed by the command will show the manual pages for the command.

For example to get help on the pg command type:

```
man pg
```

this shows:

```
pg Command

Purpose

Formats files to the display.

Syntax

pg [ -Number ] [ -c ] [ -e ] [ -f ] [ -n ] [ -p String
] [ -s ] [ +LineNumber ] [ +/Pattern/ ] [ File ... ]

Description

The pg command reads a file name from the File parameter and writes
the file to standard output one screen at a time. If you specify a
- (dash) as the File parameter, or run the pg command without
options,
the pg command reads standard input. Each screen is followed by a
prompt. If you press the Enter key, another page is displayed.
Subcommands
used with the pg command let you review or search in the file.

To determine workstation attributes, the pg command scans the file
for the workstation type specified by the TERM environment variable.
The default type is dumb.
```

The manual pages cover more than just straight forward commands, they also hold subroutines that could have the same name as commands. Also there may be commands with the same name but different actions depending upon the aspect of the system it works on.

During the man pages it may refer to a different part of the man pages. The man pages are actually split into 8 different types. These can be accessed by putting the number before the command.

The format of this is

`man` *`x command`*

where x is the switch (see list later) and command is what your looking for help on.

Man command switches

1 User commands
2 System calls
3 Library functions
4 Devices and device drivers
5 File formats
6 Games
7 Miscellaneous
8 System and operation

The man files are all held within the man directory. This is normally /usr/man for the operating system commands and /usr/local/man for any additional commands.

Typically each command registered with man will have the following information:

i.	Name	- The title and a one line description
ii.	Synopsis	- The syntax used
iii.	Description	- Information on the function and usage of the command. This normally includes examples.
iv.	Files	- Any associated files
v.	See also	- Any related information (other man pages)
vi.	Bugs	- Known Bugs or odd behaviour experienced

If you don't know what the name of the command is then you can use the -k option to find the commands that perform a certain function. For example if you need to find a directory listing you could try a search on the word list.

```
man -k list
```

One of the pages shows the following [my highlighting]

```
lindex (n)           - Retrieve an element from a list
linsert (n)          - Insert elements into a list
list (n)             - Create a list
listalias (1)        - list user and system aliases
listbox (n)          - Create and manipulate listbox widgets
listen (2)           - listen for connections on a socket
llength (n)          - Count the number of elements in a list
locate (1)           - list files in databases that match a pattern
lrange (n)           - Return one or more adjacent elements from a
list
lreplace (n)         - Replace elements in a list with new elements
ls, dir, vdir (1)    - list contents of directories
lsattr (1)           - list file attributes on a Linux second
extended file
em
lsearch (n)          - See if a list contains a particular element
lsmod (1)            - list loaded modules.
```

the ls command is the one that we were looking for.

if this is the first time this has been performed then you may be told that you have to create the whatis database. This is done by issuing catman –w . This may need to be done by root to have the appropriate permissions.

```
$ man -k list

man: 0703-310 file man not found.
  Create the whatis database using the <catman -w> command
```

Another command similar to using man with the -k option is apropos. To search for a command that performs a list function try:

```
apropos list
```

This needs the whatis database to have been created which can be achieved with the command:

```
catman
```

Another way of obtaining help for what a command is or what it does is to use the help switch. The most common switch is --help, other switches in common use include -? and -h
For example

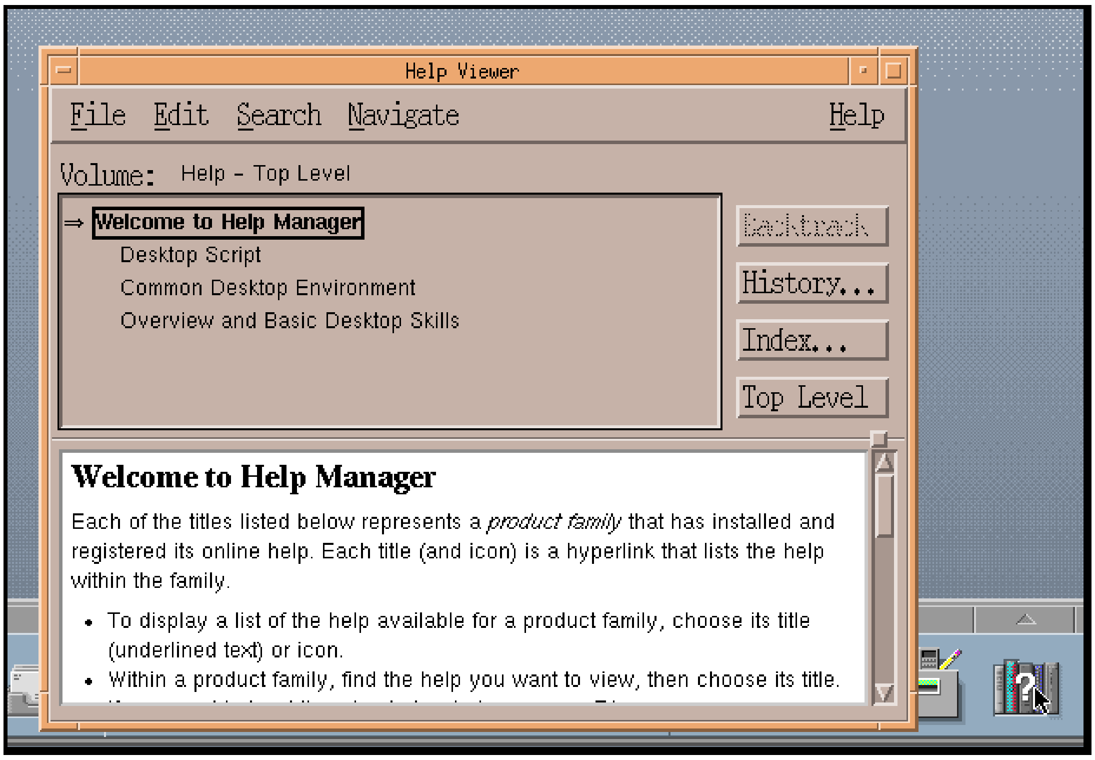
```
pg --help
```

would show the command options available for the pg command. The --help option is not necessarily as comprehensive as using the man pages. Often the --help switch will give a list of the available options only, compare this to the man display earlier.

```
$ pg --help
Usage: pg [-Number] [-p String] [-cefns] [+LineNumber] [+/Pattern/]
 [File...]
```

The --help option is the older standard that is usually implemented. However some commands support a -h option instead.
Most commands will support both the above however if one doesn't work you could try the other, which may work.

If you are running CDE there is also a further source of help in the form of the CDE help viewer (dthelpview). This is launched by clicking on the icon indicating books and a question mark normally in the bottom right hand corner next to the Trash Can. If you get an error message saying that no applications have registered with the viewer then you may need to install the CDE help files. This can be done using the easy install within smit, using the standard install disks, select Personal Productivity and then CDE Help Files.



⇒ Welcome to Help Manager
Desktop Script
Common Desktop Environment
Overview and Basic Desktop Skills

Welcome to Help Manager

Each of the titles listed below represents a *product family* that has installed and registered its online help. Each title (and icon) is a hyperlink that lists the help within the family.

- To display a list of the help available for a product family, choose its title (underlined text) or icon.
- Within a product family, find the help you want to view, then choose its title.

CDE Help Viewer the cursor is over the icon used to launch it

You can also get more help from the Solaris Manuals. These are available from the Sun Website:

http://docs.sun.com

Finding Files and Commands (find)

Sometimes you may have an idea what a command is called or the name of a file, however not whereabouts it is on the disk. The find command will scan the directory structure looking for files that match a certain search requirement.

The command is specified by specifying a directory and then searching all directories underneath that.

The most common way of using the find command is to search for a program or file by it's name. To search the entire computer for a file called filename the following command would be used.

```
find / -name filename
```

Further information can also be searched for such as the creation date of the file or it's file owner.

A few useful commands

Here are a few commands that may come in handy.

Clear

Typing clear from the command line will clear the terminal screen. This is useful if you've completed a command that has left a lot of text on the screen and you'd like it cleared so that you can see what happens with your next command.

Echo

The echo command just echo's whatever you type onto the screen.

```
$ echo This is a message to show on the screen
This is a message to show on the screen
$
```

Whilst this command might not appear to be very useful when combined with other commands via pipes or used in scripts it can indeed be very useful.

Chapter 5

Files and Directories

This chapter explains how data is stored on the disks. Normally data is stored in files, which are then contained within a directory structure. This makes finding the files easier and to make it easier to understand the purpose of a file.

What are Files and Directories?

A "classic" file is a collection of data located on a portion of a disk. The reason that I say a classic file is that in UNIX files are often used as a representation of a device or to refer to parts of the operating system. There are 3 types of files

- **Ordinary** - This contains data. One form of file is a text file however this category also includes files created by applications or indeed the applications themselves.
- **Directory** - This is a special kind of file that acts like a table of contents allowing the organisation of files (see later in this section).
- **Special Files** - UNIX operating systems allow you to access logical devices by creating virtual files that refer to them. These are found in the /dev directory. For example /dev/tty1 refers to a TTY screen, /dev/cd0 refers to a CD-ROM etc. One particular file is /dev/null which is a pointer to nothing. You may wonder why you would want a file that didn't point at anything, however this can be useful where you get an output that is not needed. Redirecting the output to /dev/null will cause it to disappear.

The number of files on a computer can be in the thousands. To have these in one place would make it very difficult to find what you wanted. Searching a list of thousands of files would be equivalent to searching for a needle in a haystack. The files are therefore collected together by commonality. These are put into groups called directories. To make it even easier these are sorted into a hierarchical tree. Moving down the hierarchical structure allows you to find more specific files.

The files and directories are referred to by filenames. The filenames can be quite long (up to several hundred characters with some implementations). They can contain any letters, digits and punctuation (including spaces). It is however better to avoid any special characters and especially spaces. This is particularly important in a command line environment like UNIX where it is necessary to interpret the differences between a single filename with spaces and two separate parameters. See creating a new file for more rules on filenames.

The directory structure starts at the root directory which is known by a forward slash '/'. Each directory below the root has it's own name and is separated by the forward slash '/'. Below the root directory are a number of "Top level Directories" Common top level directories include home; usr; var; tmp and etc

These are used for the following purposes:

home - This directory is used as a starting point for user’s home directories. These are directories that are given to users for them to store any files they create. The users directory is a subdirectory to home normally the same name as the username. e.g. /home/stewart

usr - This directory is used to store installed user applications.

var - This is used for files of variable length. Files that change a lot such as log files would be kept here.

tmp - If used this directory is used for temporary files that are only needed for a short period of file. Some systems use housekeeping jobs to periodically delete files in this directory so important files should not be stored here.

etc - This directory holds many of the configuration files used by applications.

bin or sbin - Whilst you may see bin or sbin directories at the top level these are normally symbolic links to the directories /usr/bin or /usr/sbin respectively. The bin directory carries executable programs that can normally be run by anyone, whereas the sbin directory is for programs intended for the superuser only.

lib – This directory holds additional binary files used by programs. It is actually a link to /usr/lib

export – This directory is used by NFS when configured as an NFS server

opt – Used to install programs. The name refers to optional programs, although effectively applications (or the installer) can choose to install in either /usr/local or in /opt. The opt directory is normally used for complete applications and /usr/local for smaller command line programs.

There are some other top level directories that are used as starting points for other directories.

dev - This is used to hold the devices available on the system. For example the first hard disk (hdisk0) can be referred to as /dev/hdisk0

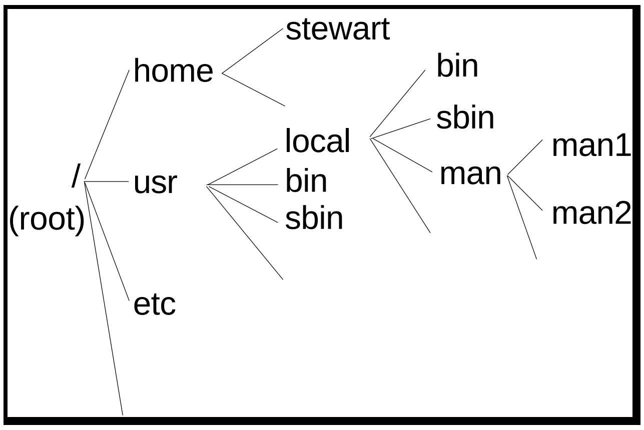
devices – This directory holds the actual device drivers. This is different to /dev which provides an easier to use logical representation of the devices.

mnt - This is the mount point which is used as a convenient place to attach other devices such as floppy disk drives and CD-ROM Drives. For example the CDROM drive may be mounted as /mnt/cd0/ although the name does not have to be the same as the device name.

proc – This is a special case directory holding information about the running system.

kernel – Another special directory used to represent the running kernel.

These then subdivide into further directories, the are known as subdirectories. The directory structure can be represented as a tree.



The above tree is far from complete there can actually be over a hundred directories on a UNIX installation.
The full directory path name starts with the root directory (/) and move up the tree with each directory separated by a forward slash. For example the man2 directory would have a full path of
`/usr/local/man/man2`

Relative Directories

So far we've been looking at the directory name in full this is known as the "absolute directory". As well as being able to specify the directory in full (starting with root) it's also possible to specify a "relative directory". To specify the relative directory start from the current position and specify the rest of the directory name. The difference between an absolute and relative directory is that an absolute directory always have a '/' at the start of the path whereas a relative directory will not. If already positioned at the directory /usr/local the man2 directory can be referenced by man/man2

Special Directories

There are 2 special directory names that apply to all the subdirectories. The current directory is marked by a single dot (.) whereas the further up can be referenced by a double dot (..) .

The man2 directory can be referenced from the /usr/local/bin directory with the relative path ../man/man2

Moving about the directories (cd)

When first logging onto a UNIX system you will normally be in your "home" directory. This is normally /home/username (for username stewart that will be /home/stewart).

You can check the current directory at any time by issuing the pwd command (this stands for print working directory).

You can move around the directories by using the cd (change directory command). To change directory use cd followed by either the absolute or relative directory (as detailed earlier).

Here's a few examples of how to change directory

Command	**What it does**
cd	Moves to your home directory. Normally /home/username
cd /	Moves to the root directory
cd /usr/local/man	Moves to the man directory (absolute path) this will work no matter what directory you are currently in.
cd local/man	Moves from the current directory (relatively). If you are currently in the /usr directory this command is identical to the above command.
cd ..	Moves up a level. If you were at /usr/local/man you would now be at /usr/local

Listing the contents of the directories (ls)

So far we have moved around the directories, however we have not looked at the files stored, or indeed seen how to find out what directories exist. This is done using the ls command.

Typing ls on it's own will list all the files and directories contained within the current directory.

```
$ cd /home/stewart
$ pwd
/home/stewart
$ ls
docs         readme.txt
```

Here I've moved to the usr directory. Displayed the current working directory and then listed the contents of the directory. In my home directory is one file and one subdirectory. You cannot tell for certain from the view above (although you could guess from there names) which is the file and which is the subdirectory.
One way of telling is to use ls with the -l option which shows a lot more information.

```
$ ls -l
total 636
drwx------   2 stewart  users        512 Sep 16 17:42 docs
-rw-------   1 stewart  users        124 Sep 16 17:26 readme.txt
```

Using this display you can tell that docs is the directory.

As far as UNIX is concerned a directory is just a special type of file so is displayed along with the other files. From the mode display (this is the 10 characters at the left of each file displayed) the first character shows what type of file it is. The remaining 9 characters show the file permissions however these will be dealt with later.
There are seven different file types, which are listed in the table below.

File Type Letter	**Description**
-	Regular File
d	Directory
l	Symbolic Link
b	Block Special File
c	Character Special File
p or s	Other Special files (not covered)

A regular file could be an application file or a text file etc.
A directory is a file that holds other files (including other directories)
A symbolic link allows a file to be redirected to another.

A block and a character file will be explained later.

We can also see that docs, is the directory by going into it and displaying the contents.

```
$ cd docs
$ pwd
/home/stewart/docs
$ ls
file1  file2
```

Here's an explanation of the Listing format shown earlier with the ls -l option.

```
    (1)     (2)  (3)     (4)        (5)      (6)        (7)
drwx------   2 stewart  users        512 Sep 16 17:42 docs
-rw-------   1 stewart  users        124 Sep 16 17:26 readme.txt
```

1. File / Directory and permission bits
2. Link Count
3. Username of person who owns the file
4. Group name for which group privileges apply
5. Character count of the entry
6. Date file was last modified
7. Name of the file/directory

The permissions of the file will be explained later.

Referring to files within a directory

A file can be referred to by just it’s filename, it's absolute directory and filename, or by a relative directory and filename. For example file1 can be referred to in the following ways.

```
From /home/stewart/docs file1                         (filename)
From anywhere           /home/stewart/docs/file1      (Absolute)
From /home/stewart      docs/file1                    (relative)
```

There is also another way of referring to the current directory using a single dot '.' . This is particularly important when running a program. Some programs that haven't been installed in the standard program directories need to be run by specifying it's directory and filename. The directory can be absolute or relative however when the file is in the current directory the filename alone is not sufficient. To get around this the file is referred to using the dot directory. For example is there is a program called exec in the current directory this would be run by using ./exec

Hidden Files

Whilst you can normally see the files in a directory using the ls command it is also possible to hide some. The normal reason for hiding a file is so that curious users don’t edit or delete the file by mistake.
To hide a file all that is needed is for the name to begin with a period ‘.’
For example the file .profile is normally hidden from view. To view a hidden file use the –a option when using the ls command.

Lost + Found

There is a special directory name called lost+found. Sometimes when the system crashes it may "lose" a file. If this happens when the system runs it's checks it will store them in the lost+found directory so that any information held in them can be recovered.

Chapter 6

Working with files

We have already covered some commands used to display the contents of a directory. This chapter covers some of the commands that can be used to create and modify files and directories.

Making a new directory (mkdir)

New directories can be created using the mkdir command. Directories can be created either singularly (in which case all directories above it must already exist) or several at a time allowing parent directories to be created at the same time.

Creating a single directory from the current directory is done by mkdir followed by the name of the new directory. For example:

```
mkdir newdir
```

The file directory could also have been created using a full path however all the previous directories must already exist. If the previous directories don't already exist then using the -p option will also create the parent directory. For example:

```
mkdir -p /home/stewart/newdir1/newdir2
```

Making a new file (touch)

A common way of making a file is to save to a file from an application (see the vi section for details of using a text editor). It is also possible to create a blank file using the touch command. The format is to have the filename after the touch command. For example:

```
touch newfile
```

The following rules should be followed when creating a file:

- The name should be descriptive of the comments
- Only use alphanumeric characters i.e. Uppercase, lowercase, numbers, #.@-_
- Although spaces can be included in a filename it is strongly advised against this
- Should not include shell metacharacters (characters used by shell pipes etc)
  *?<>/&;:![]$\'"
- Should not begin with + or -
- Should not be the same as a system command
- Filenames starting with . are hidden (cannot be seen from an ls without the -a option)
- Less than 255 characters

Whilst this is often the most common use of the touch command it has other uses as well. For example the touch command can be used to change the creation date or other properties of a file.

Removing a directory (rmdir / rm)

The safest way to delete a directory is to first remove all files from the directory. Then check the directory is empty by issuing ls -a (the -a will show all files even if they are hidden). Then change to the directory above and type

```
rmdir dirname
```

The rmdir command will only work when the directory is already empty which provides a little protection against accidentally deleting files.

For a more cavalier way of removing a directory or multiple directories the rm command can be used. To do this type rm -r followed by the directory name. For example:

```
rm -r dirname
```

Warning: This is a very dangerous command.
The rm -r command will remove all directories below the one specified.
Be especially careful if logged in as root as this can delete all the data on the disk.

There is no way of undeleting files that are deleted by mistake.
The only way or restoring the file if deleted accidentally is to copy it from the latest backup disk (if available)!

Removing a file (rm)

Files are removed by using the rm command. To delete the file type rm followed by the file name (can include a path).

There is no way of restoring a deleted file (except from a backup copy) so be careful when deleting files. A slightly safer option is to use the -i option, which will prompt before deleting each file. This is particularly useful when using wildcards (see later). It is possible to setup your profile so that you always use the -i option (see alias).

Moving / Renaming a file or directory (mv)

Files or directories can by moved or renamed by using the move command (mv). To move a file enter the mv command followed by the filename and then followed by the new directory. To rename a file or directory enter the mv command followed by the old name and the new name.

You can use the path along with the filename.

Using the -i option will prevent accidentally overwriting an existing file.

Copying a file (cp)

Files can be copied using the cp command. Enter cp followed by the existing file and then the new file (can include paths). Putting the 2nd parameter (new file) as a directory name will copy the file to another with the same name in the new directory.

Using the -i option will prevent accidentally overwriting an existing file if it exists. Wildcards can also be used to copy multiple files into different directories.

There is also a cpio command that is more versatile and allows the copying of directories and files within them. View the man pages for more information on the cpio command.

Viewing the contents of a text file (cat)

You can view the contents of a text file using the cat command. Whilst cat may seam a strange name for a command the view a file, it's because the command can be used to combine two files into a single one (the name comes from concatenate).

To view a file using cat

```
cat filename
```

you should not use this on a binary file as the output can do strange things, including sounding the speaker or accidentally re-mapping the keys.

Other commands that can be used including more, pg and view (see later). Cat is useful for automating a task as it does not need any interaction from a user.

Viewing the beginning / end of a text file (head / tail)

Sometimes you may want to look at the first few or last few lines of a file. For the first few lines you may need to look at the top of a file and take a different action depending upon the type of file, An example of needing to see the last few lines would be a sequential logfile where you want to view the last few entries rather than having to go through all the entries to reach the bottom.

The commands to do these are the head and tail commands for the top and bottom of the file respectively. The commands have a number of options allowing you to specify a certain number of lines or start from a particular place in a file however the easiest way to use it is just to enter

```
head filename
```
or
```
tail filename
```

which will display the first / last 10 lines of a file.

One use of the tail command is to look at the end of a file that is in the process of being updated. The command could be repeatedly run against a file to show any new additions to the file. If however the file has not been changed then the previous output will be redisplayed or if the file has changed too much then some lines may be missed. The better way is to use the -f option, which will display any new lines that are added to a file.

```
tail -f filename
```

Checking the type of file (file)

If you try and view the contents of a none text file you will often get garbage on the screen. Sometimes this will actually prevent you from using that terminal by changing the mapping of the keyboard or the screen. It is therefore a good idea to check what an unknown file is before viewing the contents.

One way of doing this is using the file command. Entering

```
file filename
```

will tell you if the file is a text file, a command file or a directory etc.

Printing a file

To put a file on the print queue use the following command.

lp filename

You may need to specify the printer (see man lp for more details). If you are using an X-Windows application then there is normally a print function included within the application and this could be used instead.

File Permissions

UNIX systems are designed to have multiple people using them, and as such there are controls on what each user can be done. So a user may create a file on the system that others can view, but not change, or that only certain people can see. You cannot however prevent the root user from accessing the file as they have full permissions to all files on the system.

File permissions are split into 3 different categories. These apply to:

user - the owner of the file
group - a group of people, e.g. a project team or department
others - anyone else that has a login to the computer

these are then split into 3 different permissions, that of being able to:

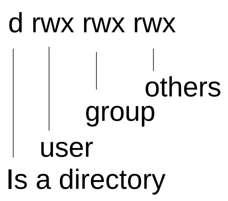
read - Look at the contents of a file / find out what files are in a directory
write - Change or delete the contents of a file / create or remove files in a directory
e**x**ecute - Can execute (run as a program) a file / can change to the directory or copy from the directory.

These are represented by the letters r, w and x respectively.

These can be seen by using ls –l on a file:

```
/home/stewart/test $ ls -l
total 14
drwxr-x---   2 stewart  stewart      512 Jan  7 15:13 dir1
-rw-r--r--   1 stewart  group01       27 Jan  7 15:14 file1
-rw-rw----   1 stewart  group01       31 Jan  7 15:14 groupedit.txt
-rw-rw-rw-   1 stewart  stewart     3502 Jan  7 15:15 public.txt
```

The first few columns represent the file type and permissions:



If the entry is filled in then it has affect if it is dashed out '-' then it does not apply.

There are also further permissions that can be set, however these are more advanced and are outside of the scope of the user guide.

Changing File Permissions (chmod)

Assuming that you are either the owner of the file or root it is possible for you to change the permissions of a file to either add or remove permissions. This is done using the chmod (change mode) command.

The chmod command can be used in one of two ways. The Symbolic Format or the octal format. Symbolic is useful for new users as it is easier to use, however once the octal format is learnt it can be a powerful and quick way of changing file permissions.

The basic format is

chmod mode *filename*

It is only the format of the mode parameter that is different.

In symbolic format permissions are added or deleted using the following symbols:

u = owner of the file (user)
g = groups owner (group)
o = anyone else on the system (other)

+ = add permission
- = remove permission

r = read permission
w = write permission
x = execute permission (if the file is a program then this gives permission to run it)

For example from the previous ls entry we have a file called file1 which we would like the group to be able to write to:

```
-rw-r--r--   1 stewart  group01       27 Jan  7 15:14 file1
```

Using symbolic format we just need to add write access to the group which can be done using:

```
chmod g+w file1
```

In Octal format the mode is based upon a octal number representing the different mode permissions, where each of the permission groups (user, group, others) has an octal value representing the read, write and execute bits. This requires a little bit of knowledge on binary and octal number bases.

To use the octal format we don't just put the changes, but issue the command on the entire permissions for the file.

	User	**Group**	**Others**
Symbolic	rwx	rw-	r--
Binary	111	110	100
	4+2+1	4+2+0	4+0+0
Octal	7	6	4

The file permissions would therefore have the octal number 764 and would therefore be changed using the command

```
chmod 764 file1
```

A basic way of working this out is to add the following numbers depending upon the permission required.
Read = 4
Write = 2
Execute = 1

Therefore if you wanted to set read to yes, write to no and execute to yes, this would be 4+1=5

Changing the file owner (chown)

It is possible to change the owner of a file however normally this requires the user to be logged in as root. The owner or group can be changed using the chown (change owner) command.

The format is as follows:

```
chown user:group filename
```

Chapter 7

Making the Most of UNIX Commands

This chapter explains how different commands can interact to provide extra functionality. It introduces a number of ways of dealing with a programs output. It will introduce some new programs to make it easier to perform some tasks.

Whilst the number of options on each UNIX command may seam overwhelming at first this is part of what makes UNIX so powerful. Another of the features that makes UNIX so powerful is the ability to combine several commands to make them more useful. This can be achieved either by stringing commands together on the command line or by bundling the commands together into a script file which can range from something very trivial to a program in it's own right. I will not discuss scripts further in this book (see volume 7) however I will show how multiple commands can be combined.

Using command switches

The most basic way of extending the functionality of a command is to use some of the switches available.

These can be found using the man command as mentioned in an earlier section.

The simple example of this is the ls command. Without any switches all that is displayed is a list of all the files in the current directory.

```
$ ls
docs         readme.txt   smit.log     smit.script
```

By adding the '-l' option more information is provided.

```
$ ls -l
total 636
drwx------   2 stewart  users        512 Sep 16 17:42 docs
-rw-------   1 stewart  users        124 Sep 16 17:26 readme.txt
```

The pipe command (|)

There are a number of features missing from some of the commands. For example there is no way of viewing the output a page at a time or of sorting the output into a certain order. Whilst this may sound like a large disadvantage at first it is not really a problem as there are a number of commands that are specially designed to perform those exact functions. By restricting these tasks to other commands it makes the commands simpler and the number of options down.

The way this is achieved is by 'piping' the standard output into another command that is specifically designed to perform the certain function. The pipe command is a vertical bar '|'. This is normally found at the bottom left of the keyboard and is typed by using shift and the '\' key (this assumes a keyboard with the UK layout).

The first command is entered first followed by the pipe command and then followed by the second command. Any output from the first command is then used as input to the second command.

For example to sort a basic directory listing by name the ls command is piped through the sort command.

```
ls | sort
```

The output can be redirected through any number of pipes each one changing the output in someway. The full command is referred to as a pipeline.

Redirecting stdout, stdin and stderr (> <)

The output from a command goes to the standard output (stdout), which is normally the screen. Whereas the input is normally taken from standard input (stdin), which is normally the keyboard. To automate processes it is sometimes required to change this so that for example the output from a command is routed to a file or the printer. This is done by redirecting the stdout and stdin streams.

The output from the ls command could be redirected to a file in this case called dirlist.txt

```
ls > dirlist.txt
```

If the file dirlist.txt already exists it will be deleted. It is also possible to append the output to the end of an existing file by using >> instead of >.

For example

```
echo "This is the next line of the log" >> log.file
```

Whilst you may see all the output from a command on a single screen this is not all necessarily coming from stdout. There is also another data stream called standard error which by default is directed to the same screen as stdout. This data stream is used to send messages regarding any error messages. The advantage of having this as a separate stream is that even if you redirect stdout, because you are not interested in the output or just want it for reference you will instantly see any error messages on the screen. If this is a command running automatically without user interaction then there will not be any one to see messages put on the screen. The standard error data stream can therefore be redirected the same as stdout by prefixing the redirect by the number 2 digit. In fact the stdout data stream should be prefixed by the number 1 digit, the 1 is normally dropped to save typing. To therefore redirect any error messages to an error.log file and the normal responses to a log file the following would be used.

```
command >log.file 2>error.log
```

The single >'s can be replaced by double >>'s if you would like the output to be appended to the file rather than to overwrite the file.

It is also possible to write both stdout and the standard error stream to the same file. This is not simply a case of using the same file name in the above command as you might expect. The reason for this is that a file can only be opened for writing by one process at a time. The two redirects are two different processes and would not allow both streams to write to the same file. This can however be achieved by redirecting the error data stream to the stdout data stream using 2>&1. Which now gives:

```
command >output.file 2>&1
```

In a similar light you cannot use a file used as input to the command to redirect the output to. For example it is not valid to issue the following command

```
sort file1 >file1
```
This is not valid

instead the output would have to be redirected to a temporary file and then renamed to the required name.

```
sort file1 >/temp/tmp$$
mv /tmp/tmp$$ file1
```

The file ending in $$ will actually be created by the system with a unique number. This is useful for temporary files as it prevents you overwriting a temporary file in use by a different process.

The use of stdin, stdout and stderr is possible using only the single less than / greater than signs because of the way that processes are assigned to a file descriptor table.

The file descriptor table is a list of numbers relating to open files. The first 3 files to be opened are stdin, stdout and stderr, these are numbered 0 for stdin, 1 for stdout and 2 for stderr. Therefore stdin and stdout can be referred to by < and > respectively (no further filename) whereas stderr requires 2> to ensure it is output stream numbered 2 that is to be redirected.

These redirects are fine for use in batch programs that are run without anyone monitoring the system. Sometimes it is necessary for someone to monitor the output of the command but also for it to be duplicated into a file for logging purposes or to perform further processing on.

The tee command is used within a pipeline and whatever input it receives it both puts into a file and forwards on the pipeline.

```
command | tee file1
```

The line above will take any output from the command and put a copy in file1 as well as sending it to the screen. This could be combined further by allowing the output to be further processed by another command. The tee command can be used any number of times in a pipeline like this.

```
command1 | tee file1 | command2
```

If you want tee to append to a file rather than overwrite it the -a option is used.

The same basic redirect can also be done in the reverse direction in that an interactive program that requires input from a user can be automated. For example with an interactive program such as ftp (file transfer protocol). The ftp program allows files to be transferred from one computer to another over a network. This however needs a user to type the commands in to transfer the file. Instead the commands should be entered into a text file the same as how they would be entered from the keyboard. The file is then directed into the program in place of the stdin.

```
ftp rs6k.mynet.com <commands.txt
```

Redirecting to a file can have an unfortunate consequence if the file already exists and does not want to be replaced. By using the redirects incorrectly it is possible to accidentally overwrite an important file. In the korn shell there is an option that allows us to prevent overwriting files by mistake. This is the noclobber option and is set by typing

```
set -o noclobber
```

If an attempt is now made to overwrite a file the shell will issue an error message and prevent the file being written to. The no clobber option can be turned off using

```
set + noclobber
```

If required the noclobber option could be put an a users .profile to have this set automatically.

Chapter 8

Managing Your Account

This chapter provides information useful to having your account work how you want it. It introduces some of the user settings, but assumes that you don't have superuser privileges. For more detailed information on managing userid's you should consult the User Administration part of the System Administrators book.

Whenever you login to a UNIX machine using your username then certain things will be automatically setup. These will be different for different users on the system and to an extent can be customised by you. I will explain where some of the settings originate from, even though you may not be able to change them. At least then you will appreciate why the machine is behaving as it does.

Normal Vs’ Root Username

You may come across references to the following terms: root username, superuser and userid 0. These all refer to the same thing, which is basically a user with full privileges to your system. Anyone logged in as that user can read and update any file on the system, they can read any emails and perform operations as though they are any user. For security reasons the root password should therefore be well guarded. If you are in the position of owning the root userid (i.e. this is a machine that you own and manage) then you might think that it is a good idea to login as root then you don’t have to worry about not being able to do anything. This is not a good idea as it also means that any mistakes that you make could have bad consequences for the whole system. For example the rm command is used on a regular basis to delete unwanted files. Using the –r option whole directories can be deleted at once. If you ran
rm –r *
from the root directory (a command often run, but **never** from the root directory) then as root you would delete all the files on the system, however if you were only a normal user you would not be authorised to delete anything from the root directory and so the command would fail. It is therefore better to perform most functions as a normal user, and only use the root userid when it is necessary to use the elevated privileges.

It would be quite tedious to log out and back in again every time you wanted to change userid. There is however a command that allows you to switch userid without having to logout of your existing session. The command is called su. It originally stood for “SuperUser” as all it allowed was to change to the root userid. With modern versions (meaning within the last 10 years so almost all versions in current use) you can change to any user and not just root, it is therefore more appropriate to call it the “SwitchUser” command. For obvious security reasons you need to enter the password for any user you are su’ing to unless you are root. As root has the ability to reset or change the usernames it has the unique ability of being able to su to any other userid without giving a username. The su command is discussed further in the Systems Administrator Book.

Shell Variables

Whilst you are in the shell there are a number of variables that the shell needs to be aware of. These could typically hold information about you, about the shell you are running in, or about your personal preferences. For example the shell needs to know what your home directory is, so that you can easily change directory or save files. It stores this in the $HOME variable.
Another variable commonly used is the $TERM variable. This describes the type of terminal you have so that the shell or applications know how to send control the

screen. For example a vt100 terminal may have a different command to clear the screen than is used by a Wyse Terminal.
There is another one that you may need to change which is the $PATH variable. This indicates which directories the shell will look in to find a program that you've asked to be run.

Normally all system defined variables are in uppercase whereas user variables are in lowercase.

You can list all variables by using the set command.

e.g.

```
$ set
HOME=/export/home/stewart
IFS=

LOGNAME=stewart
MAIL=/var/mail/stewart
MAILCHECK=600
OPTIND=1
PATH=/usr/bin:/bin:/usr/sbin:/sbin
PS1=$
PS2=>
SHELL=/sbin/sh
SSH_CLIENT=192.168.2.3 62880 22
SSH_TTY=/dev/pts/5
TERM=ANSI
TZ=GB
USER=stewart
```

One instance that you may need to change a setting is if you connect to the Solaris machine by using telnet from a Windows 9x machine. If you use the standard telnet client then you will probably end up with a terminal type of ANSI (see TERM in above screenshot). Whilst you may be able to issue commands normally many full screen applications will not work. See the following error when trying to use the vi editor:

```
$ echo $TERM
ANSI
$ vi test.txt
ANSI: Unknown terminal type

[Using open mode]
Segmentation Fault
$
```

The echo command is used to display the value of $TERM to confirm what it is defined as.

The rather cryptic error message is basically saying that it cannot run because it does not know how to control a terminal defined with type "ANSI".

To overcome this you first need to change the value of the variable. The windows telnet command supports ANSI/vt100 under the same profile, therefore we can try the vt100 terminal type which is better supported by UNIX. The following shows what the effect of changing the value is:

```
$ TERM=vt100
$ echo $TERM
vt100
$ vi test.txt
ANSI: Unknown terminal type

[Using open mode]
Segmentation Fault
$
```

Note that I have used the correct command to change the TERM variable (ie. TERM=vt100). The $ sign is not used when changing the variable.
The echo command confirms that it has been changed however the application still failed.
Looking at the error message he problem is that it does not know about the ANSI terminal type. But haven't we just changed it to vt100, why is it complaining about us having a ANSI terminal defined? Either I'm lying or the program is still trying to use the old value.

The reason that this doesn't work is that the value changed is the variable being used by the current shell we are using (remember this is a program normally either sh or ksh). When we start a new program, or indeed a new shell it runs as a child and does not have access to the variable changed in the new shell. See the following example:

```
$ echo $TERM
vt100
$ ksh
$ echo $TERM
ANSI
$ exit
```

Here I have displayed the $TERM variable (vt100). I then started a new child shell (ksh) and then displayed the $TERM variable again. Inside the child shell the variable is now back to the original ANSI. A similar thing is happening whenever we run a program. Fortunately there is a command that lets you set a variable so that it is passed to any new child shells. The command is export. Now we try the commands again:

```
$ TERM=vt100
$ export TERM
$ vi test.txt
```

This time the vi program runs successfully. If you run the ksh command again and echo the $TERM variable you will notice that that too has changed to $TERM.

You can get rid of a variable by using the unset command. It is better not to remove any system-defined variables (unless you really do know what your doing), however you may have created your own variable that is no longer needed. The following example creates a new variable before removing it.

```
$ MYVARIABLE="UNIX is great"
$ echo $MYVARIABLE
UNIX is great
$ unset MYVARIABLE
$ echo $MYVARIABLE

$
```

Below are a list of some other variables you may come across:

- **LOGNAME** - This holds your login name for use by certain commands. This is one variable that cannot be changed.
- **MAIL** - This holds the name of the file where your mail is sent
- **MAILCHECK** - How often the shell will check to see if you have new mail
- **TERM** - The terminal type you are using / emulating
- **umask** - This determines what permission bits will be set when a new file is created. This is a mask so an inversion will give you the default file settings. E.g. the default of 022 will create a new directory with 755 (user can read, write, execute everyone else can read, execute). When a regular file is created the execute bit is not set so you would get 644. You may want to make this a little more secure by setting to 027 (don't give any permissions to other users) or even 077 (only give owner permissions). See the book on security for more details.
- **PATH** - This is a colon separated list of all the directories that will be searched to find a command typed in by the user. Some people add a :. to this string to say if the command is not found in the rest of the path then try the current directory, however there are security issues associated with this (see security section) and certainly this should not be set for the root user.
- **PS1** - This is the system prompt and is set to a $ by default. You may want to change this to include your current directory name using the command:
  PS1='$PWD $'
- **ENV** - Not included by default it can be useful to add this to your .profile. To set the shell environment file to .kshrc (this is the normal name for the Korn Shell) you would include the following line:
  export ENV=$HOME/.kshrc
  This is especially useful if using a X-Windows where the .profile is not called when you start a new xterm.

.profile

The previous chapter explained how to change your $TERM variable so that you can run full screen programs when logging in from a Windows 9x machine. Whilst the commands were straightforward enough you can probably appreciate it would be awfully annoying to have to type that in every time you logged in. There are several places that these variables can be defined, however there is one that is run every time you login that you are allowed to edit yourself. This file is called .profile and is located in your home directory (the directory stored in $HOME).

To add anything just use your preferred editor (the vi later is explained in a later chapter). If the file does not exist then the editor will normally create a blank file on your behalf. To automatically change the Terminal type to vt100 enter the following lines.

```
TERM=vt100
export TERM
```

The problem here is that you may logon to the computer using different methods (physically on the machine; from a windows machine; from a dumb terminal located in a different room etc.). If you do then you may find that your choice of terminal on one screen may conflict with another. The following lines show how you could add an entry that would only select vt100 if you connect with an ANSI terminal (ie. Windows Telnet).

```
if [ $TERM="ANSI" ]
then TERM=vt00
export TERM
fi
```

I will not explain this here; just accept that it works. The if statement is explained in volume 7 "Unix Scripts".

Another common complaint with Solaris users is that the "Backspace" key doesn't work as expected. This can be a common frustration but you can get around it by running the following command.

```
stty erase ^H
```

Here the ^H is generated by pressing the backspace key.

You could add this to the .profile however you may find that when you press the backspace key to generate the ^H it will actually work as a backspace key (just when you don't want it to). To get around this then enter the following command (note: do not enter the stty command prior to this as it will prevent you being able to create the ^H).

```
echo “stty erase ^H” >> $HOME/.profile
```

The echo command displays the entered line, but it is then redirected onto the last line of the .profile file. Then when you login the backspace key will work as expected.

Note that anything added to the .profile file will not take effect until you logout and then login again.

.kshrc

This file is similar to the .profile file however is run whenever the Korn shell is started rather than when you login. Again this is located in the $HOME directory.

When using X the .profile is loaded when X-Windows is started. This sets up the system wide variables, however there are commands that can be set that are specific to the shell and not to the overall session. Normally these will not be passed onto the virtual terminal unless they are put in the .kshrc file.
The name of the file is not important, however it must be the same as the ENV variable set in the .profile file. It is not normally in the .profile by default (unless your system administrator has set it up that way), so you need to add the following line in your .profile file first.

```
export ENV=$HOME/.kshrc
```

An example of a command that would work differently if it is in .profile and .kshrc is

```
set -o vi
```

including the above command is the same as starting the shell with

```
ksh -o vi
```

which sets the shell environment into the vi style input.

If this was in the .profile then whenever a login was made into a terminal it would take effect, however when starting a terminal from X it is not a new login and therefore would not be invoked. If however this is included in the .kshrc then whenever the Korn Shell was started it would be invoked and as the Korn Shell is invoked by both a login and a new terminal started in X this would happen all the time.

Some of the commands that could be put into the .kshrc file.

- **set -o vi** This command sets the environment into the vi style input. What this does is to provide a level of command recall for the shell. To really understand this you need to understand how to use the vi editor (see the chapter on the vi editor). To use the recall keys you first need to press the escape key (this may be mapped differently on your system however is normally the key marked Esc). Then you are in command mode and can recall commands and move around using the keys "hjkl" ('h'=left, 'j'=down, 'k'=up, 'l'=right). You can then delete a character by using the 'x' key. Pressing the 'i' key will insert before the cursor and pressing the 'a' key will insert after the cursor. Pressing either 'i' or 'a' will put you into insert mode and to return to command mode you press the escape key again. This may all sound very complex however if you are able to use the vi editor (a useful skill to acquire) then you will find this very familiar.
- **alias** - The alias command is used to set up alternative names for commands. The format of the command is:

```
alias newcmdname='normalcmdname -args'
```

What will happen then when you enter newcmdname is that it will be replaced with normalcmdname -args. Note: this will happen only on commands entered at the command line and will have no effect on script files. A few examples of where this would be useful is shown below:

```
alias rm='rm -i'
```

This will include the -i option on any delete command. What this does is provide interactive prompting so that you will be asked if you want to delete each file.

```
alias up='cd ..'
```

Whenever you enter the up command it will take you one level up the directory tree. This is useful if you, or a user have trouble remembering a command as you can make easy to remember alias names.

```
alias computer2='telnet computer2.mynet.com'
```

If you often telnet to another computer (computer2) this will save you having to type in the whole command instead of entering 26 characters you just need to enter 9 characters.

History

It is possible to recall previous commands using the –o vi option explained earlier, however it is also possible using the history command.

Issuing the history command will show the last few commands entered.

```
$ history
52      clear
53      ls
54      cd test
55      ls
56      pwd
57      cd ..
58      vi temp.txt
59      history
$
```

to recall one of the commands listed enter an r followed by the number

```
r 55
```

by default the history command will give you the last 16 commands entered.

The history is held in a file called .sh_history and can be edited using the fc command (see the man pages for more details).

Chapter 9

The vi text editor

One thing you'll need to do when you start using any UNIX system is to edit text files. Text files are used to control many of the settings of the system and many applications. Whilst there are many text editors for UNIX vi is provided on just about every flavour and installation of UNIX. If you learn vi then you'll be able to edit text files no matter what system you're on.

vi is a very powerful text editor. It runs in a standard terminal and uses the standard keyboard (it is not reliant on the arrow keys, or the insert, home, pgup keys). This makes it equally as useful on a graphics workstation as on a minimal text based terminal or indeed via a telnet terminal.

To someone new to UNIX the vi text editor can be very daunting. However learning to use vi can be one of the most useful tools that a UNIX user can know. Whilst anyone with a X-Windows system may prefer the ease of use provided by the X based editors such as dtpad or xedit you cannot always guarantee to have them available. Whereas regardless if you're working on a graphics or text only terminal almost all UNIX systems will have the vi editor installed. Also once you know the basics for vi you can use these in other circumstances, for example the korn shell can be setup to have the vi keys for command recall and edit (see details on the .kshrc file) or the same keys can be used to scroll through a file using the more command.

Rather than just describe what all the different keys do I've worked through a little example of creating and editing a text file, however it is first important to understand about the different modes of the vi editor.

Editor Modes

If you have not used vi before, then having to switch between the different modes can seam a bit alien at first, however is actually quite simple in operation. The most common two are "Insert mode", and the "Command mode".

Whenever you start vi it will be in command mode. In this state whenever you type something, instead of being included in the file it will be used as a command for the editor. Example commands include moving around a document, deleting parts of the document, and file operations such as saving the document and exiting.

To add the text that you type in at the keyboard requires you to be in insert mode. When in insert mode any text entered at the keyboard is added to the document at the current cursor position.

To switch between the different modes involves pressing the appropriate keys. To move to command mode you would press the ESCAPE key. A useful feature of the ESCAPE key is that it still works when in command mode. Indeed if you forget which mode you are in just press the ESCAPE key and regardless it will put you in the command mode.

There are several different keys that allow you to go from the command mode to the insert mode. By using the appropriate key, the number of keystrokes required can be reduced. Some of the keys are listed below:
i Insert - any text entered will be put immediately before the current letter.
a After - any text entered will be put immediately after the current letter.
A After end of line - any text entered will be put after the end of the current line.

It is also possible to run ed commands by pressing colon ':' when in command mode. The ed commands are based on command line editing used by the "ed" command line editor. The vi editor is based on the ed program that is a basic command line editor.

Worked Example with VI

Following this example will introduce a lot of the functionality of the vi editor.

Starting VI

To start the editor enter vi followed by the filename.

```
vi sjayouth.txt
```

If the file already exists then it will be opened and displayed on the screen. If it is a new file then the initial screen will be blank. The editor is started in command mode. The file opened is shown below. It is an article from a St. John Ambulance Newsletter that I wrote in September 2000:

Mention St. John Ambulance and most think of adults trained in first aid helping those in need. However St. John actually has one of the biggest youth organisations in the country, catering for all children from the age of 6 upwards.
The youth area is split into two age groups. Badgers are for children from 6 to 10 and Cadets are for members aged from 10 to 18. Both groups are open to boys and girls.

Badgers make friends and have fun. The badgers work towards 9 different badges in areas from first aid to hobbies. After gaining all 9 badges he / she can become a Super Badger.

Cadets play games and learn first aid, like the badgers. Cadets then get to practice the first aid on real casualties. There are adult members ready to give a helping hand whenever it's needed. The Cadet’s work on an award scheme leading up to the Grand Prior Award.

For more information see www.sja.org.uk

Inserting a New Line

We will now add a line before the text with the title "Young Members". The cursor is located in the top left hand corner, which is exactly where we want to insert the text. When we started the editor we were in command mode so we now need to change to insert mode by pressing:

```
i
```

We can then type in the title

```
Young Members <Enter> <Enter>
```

Pressing the enter key moved the following text to the next line and then inserted a blank line.

Moving Around the File

Now imagine that I wanted to replace the word "practice" with "use". To move around the file it may be possible to use the cursor keys. Some terminals do not support the cursor keys and the alternative is to use four of the standard keys. These are
h - left
j - down
k - up
l - right
To use the standard keys as cursors requires that you are in command mode.

To move to the word "practice" first press
ESCAPE
to change to command mode.

Once in command mode then press the 'j' key until the cursor is next to the line containing the word. Then subsequent presses of the 'l' key will move until the cursor is on they letter 'p' of practice.

Deleting Letters and Inserting a New Word

Now press the 'x' key, which will delete the letter under the cursor. Press they 'x' key 7 more times until the word has been deleted. Then press the 'i' key to return to insert mode and enter the word "use".

Save the Document

We will now save the file in case we accidentally make a mistake in future. To save the current document you need to change to command mode. Then press the colon key (:) to create a command line and enter 'w' followed by the enter key.

The file will then be saved.

Replacing a Letter

Prior to the introduction of the children's act the age for cadets was up to 16. Now we shall change the age from 18 to 16. We shall move the cursor and replace the figure 8 with the figure 6.

First press ESCAPE and use the 'k' key to move up the document up to the line containing the figure 8. Then move across the line using the 'h' and 'l' keys until the cursor is over the figure 8.

To replace the character press the 'r' key. Then pressing the '6' key will cause the number to be replaced with a 6. You will still be in command mode.

Search and Replace

For this exercise we will now replace the occurrence of 18 back to 16, however this time we will do it using a ed command. We can find the occurrences of a word by using a regular expression. So to search for 18 we would enter /18 followed by enter. This would search from the start of the file until it reached the first instance of 18.

If instead we wanted to replace 18 with 16 directly then the command that would be entered is `:%s/18/16/g`
The colon tells the editor that what follows is an ed style command. The %s is to initiate a search and replace. The first string between the '/' characters is the item to search for, the next is the string to replace with. This would work on every line of the file, but only of the first occurrence on each line. The g option makes the command work globally for every possible occurrence.

This may also replace other parts of strings such as 180 would be replaced with 160. So it is important to ensure that the command that is issued is what you really want it to do.

Save and Exit

Now we will save and exit the document. Press the ':' key followed by 'w' (to write - save) followed by 'q' to quit the program.

This is then end of this example.

Preferences

Within vi it is possible to change some of the settings. These are done by issuing a ':' followed by the set command with the option as a parameter. For example to display line numbers in the editor you would issue the command:

```
:set number
```

This could then be turned off again by using the word no in front of the options. So to turn the number option back off you would issue:

```
:set nonumber
```

Other options that can be set include:

all	Display all options & current settings
errorbells	Sounds a bell sound whenever an error occurs
ignorecase	Makes search commands case insensitive
showmode	Shows the current mode in the bottom of the screen

These settings only last as long as the current session. To have these commands run whenever vi is run then the required commands should be put in a file called .exrc in your home directory. These should be entered without the colon e.g.

```
set number
set showmode
```

Glossary

|. See pipe
&. See background process
>. See redirection
<. See redirection

A

access permission. The access allowed to a particular file. This can be based on the different levels of access (read, write, execute) at different classes of users (owner, group, all others).

ACL. Access Control List. Used to control access to a file (see access permission).

ACLST. Alternating Current Logic Self-Test

address Space. The range of addresses availaable for a processes code and data.

AIX. Advanced Interactive Executive. IBM's implementation of the UNIX operating system.

alias. The process of assigning a new name to a command

ANSI. American National Standards Institute

application. Program used to perform a certain task.

ASCII. American Standard Code for Information Interchange. Collection of character sets used throughout the computer industry.

awk. Interpreted programming language. Useful for it's pattern matching abilities and is often used in combinations with shell scripts.

B

background process. A process running independently of the initiating terminal. Started by ending the command with an &. The starting process is then no longer waiting for the "death of the child process".

backup. Having a copy of data or code used incase the original information is destroyed.

BSD. Berkely Software Distribution. A UNIX thread with differences from the base UNIX platform, although now mostly combined with other platforms.

block device. A device that transfers data in fixed block sizes. Normally 512 or 1024 bytes.

booting. Starting the computer from a power off or a system reset.

byte. Storage used to represent a character. Equal to 8 bits of data.

C

C. Programming language in which the UNIX operating system and most applications are coded in.

CDE. Common Desktop Environment. X Window Manager developed jointly between Sun, IBM, HP and SCO to provide a common feel to UNIX running on different machines.

CGI. Common Gateway Interface. A method of providing client server interactivity between a web browser and a server. CGI is implemented by scripts on the server and requires nothing special on the client. This is known as "server side scripting"

character I/O. The transfer of data byte by byte as used by slower devices, such as terminals and printers.

child. A process emerging from a fork.

CISC. Complex Instruction Set Cycles. Processors with a large number of instructions. These tend to be slower due to the number of instructions that the processor has to be able to handle

client. User of a network service.

command. A request to perform an operation or run a program. When arguments are added to the command the entire string is considered to be the command.

console. Terminal used by the kernel to send messages to.

CRC. Cyclic Redundancy Check

current directory. The currently active directory. If no directory specified then the directory that a command will act upon.

CUT. Co-ordinated Universal Time (the same as GMT)

D

device driver. Program that controls a hardware device, such as printer, disk or screen.

directory. Special file containing the names and controlling information for files and directories.

disk / diskette. Storage medium used for transferring data between machines.

DoS. Denial of Service. Where a computer is attacked so that it can no longer perform it’s role.

E

EBCDIC. Alternative character set to ASCII. Used on IBM mainframes.

editor. Program used to enter and modify text or other files.

environment. Collection of variables passed to a program or shell script by the invoking process.

escape. The \ character used to indicate that the next character in a command is normal text with no special meaning.

Ethernet. Networking protocol used to connect a LAN.

execute permission. Permission to run a program or list the contents of a directory.

F

field seperator. Character used to separate one field from the next. Normally space or tab.

FIFO. First In, First Out. Queuing process where first into a queue is the first out.

file. Collection of related data stored and received by it’s given name.

file system. Collection of files and structures on a physical or logical media.

flag. Option sent to a program.

full path name. Directory name given starting from the root (/) directory.

G

gateway. Device acting as a connector between two separate networks.

GMT. Greenwich Mean Time (the same as CUT). The time at Greenwich England, but not taking into account any changes due to British Summer Time.

group. Collection of AIX users show share a set of files.

H

hardware. The physical equipment.

heterogeneous. Applies to networks consisting of products from multiple vendors.

homogeneous. Applies to networks consisting of products from a single vendor.

I

interpreter. Program which “interprets” program statements directly from a text file.

IP. Internet Protocol

IPL. Initial Program Load - the booting sequence of a computer.

J

Journaled File System. A method of storing information on the hard disks.

K

kernel. The core of the operating system that provides the fundamental running of the system.

kill. To prematurely terminate a process.

L

LAN. Local Area Network. Network normally within a single location.

line editor. Editor that processes one line at a time.

link. Alias one directory or file to another.

Linux. Open Source UNIX like operating system.

login. To provide your identity to the system to gain access.

logout. To inform the system that you no longer need access for this session.

M

make. Programming tool that helps make a program from the source code.

memory. Storage medium, normally refers to volatile memory held in the system.

metacharacters. Characters or combinations of characters to represent different characters or sequence of characters.

Motif. Graphical user interface for X Windows.

N

NFS. Network File System

null device. A logical device used to obtain empty files or dispose of unwanted data.

O

OEM. Original Equipment Manufacturer.

operating system. Software interfacing between the applications and the hardware. It controls how applications can run on the system

owner. The person who created a file, or to whom ownership has been transfered to.

P

parallel processing. Having more than one processor in the same system.

password. Secret character string used to verify user identification.

PATH. Variable which specifies where to search for program files.

path name. Filename specifying directories leading to that file.

permission. Authority given for a set file.

pipes. System routines that can take the output from one process and feed it in as the input to another. The bar '|' character is used to indicate to the system that you wish to use a pipe.

POSIX. Portable Operating System for Computer Environments. Set of open standards for an operating system.

process. Unit of activity known to the system (usually a program).

profile. File in the users home directory executed at login to customise the environment. The actual file is called .profile

R

read permission. Allows reading of a file

redirection. The use of a different device or file instead of STDIN and STDOUT which use the symbols < and > respectively.

regular expression. An expression that specifies a set of character strings using metacharacters.

relative path name. The name of a directory or file using the directories from the current directory.

RISC. Reduced Instruction Set Computer. Processer with a small number of individual instructions. By only being able to have a small number of instructions the processer can handle them quickly and generally faster.

root directory. The topmost directory that contains all other directories in the file system.

RPM. Redhat Package Manager. Used to bundle applications so that they can be easily installed under Linux.

S

scalability. The ability for a computer to accomodate growth with the minimal of effort.

SCCS. Source Code Control System

server. A provider of service in a computer network.

setuid. A permission that allows a program to run as though started by a different user.

shell. User Interface of a UNIX operating system.

shell program / shell script. Program consisting of shell commands in a text file.

signal. Software generated interrupt to another process. As used by the kill command.

SIO. Serial I/O Register

sockets. When a network session is connected a socket is used to represent the address and ports of the systems.

software. The programs run on a system (the part of a system that is not physical).

STDERR. Standard Error. The data stream where errors are normally sent. This is normally the console although it can be redirected.

STDIN. Standard Input. The data stream where input comes from. Normally this is the keyboard although it can be redirected.

STDOUT. Standard Output. The data stream standard messages are outputted to. This is normally the terminal although it can be redirected.

subdirectory. A directory that is subordinate to another directory.

superuser. The system administrator with priviliages allowing them to access every file in the system. This is normally the root user.

swap space. A space on disk where memory can be swapped into to make space for other programs.

system. The computer and it's associated devices and programs.

System V. The thread of UNIX that retained the AT&T style rather than the BSD style.

T

TCP. Transmission Control Protocol

TCP/IP. Transmission Control Protocol over Internet Protocol.

termcap. File containing the capabilities and functions of a terminal.

U

UNIX. Multi-user Multitasking operating system. Originally developed at Bell Laboratories in the early 1970's.

V

vi. Visual Editor. A text editor used within a text terminal. Very powerful, but can be difficult to learn initially. Available on just about every UNIX like operating system.

W

wild card. Metacharacter used to specify one or more replacement characters. e.g. * allows any number of charcters to match.

window. Are of the screen in which the running program is displayed.

working directory. Directory in which the current program is running and upon where any actions (not specifiying a directory) will be taken.

write permission. Permission to change the contents of a file or directory.

X

X-Windows. Interface to the system that provides windows. Also useful in distributing applications as the application can run on a different machine to the one where the screen and keyboard are being used.

Command Summary

The following are a list of useful commands.

at ***time job***
Runs a command at a specific time.

cat ***file1***
Display the contents of the text file “file1”.

cat ***file1 file2*** **>** ***file3***
Combine the files file1 and file2 and output into file3

cd
Change to the users home directory

cd ..
Move up a directory

cd ***directory***
Change to the specified directory

chgrp ***group file***
Change group ownership for a file.

chmod [ugo][+/-][rwx] ***file***
Change permissions of a file using symbolic form

chmod ***XXX file***
Change permissions of a file using numeric form

chown ***owner:group file***
Change the owner / group of a file or directory

chown -R ***owner:group directory***
Change the ownership of subdirectories and files.

compress ***filename***
Compresses a file so that it takes up less space. Useful prior to transferring the file to another system using a network or tape. Files compressed with the compress program are suffixed with .Z

cp ***file1 file2***
Copy file1 to file2

cp -R ***dir1 dir2***
Copy a directory and subdirectories from dir1 to dir2

date
Shows and sets the system date and time.

del *file1*

Deletes a file after asking for confirmation. Ignores file protection allowing the owner to delete a file it owns.

df

Displays available space on all file systems

diff *file1 file2*

Compares two different text files and indicates the differences

du

Shows a summary of filesystem usage

e *file*

Edits a file using the INed editor

ed *file*

Uses the ed editor to edit the file.

env

Display environment variables

find *path* -name *filename*

Finds files named filename starting from directory path.

ftp *hostname*

Interactive file transfer program for transferring files over the network.

grep *pattern file*

Searches a file for the pattern

gzip *file*

Compress a file using the gzip program. This is not included with AIX as standard but is often installed by system administrators. Files that have been compressed as suffixed with a .gz

gunzip *file*

Uncompress a file that has been compressed with the gzip program. This is not included with AIX as standard but is often installed by system administrators. Files to be uncompressed will normally end with .gz

head -*count file*

Display count number of lines from a file

help

One page display of help for new users

kill pid

Terminates a process

ln *file1 file2*
Links file1 to file2

ln -s *file1 file2*
Creates a softlink instead of a hard link

ls *file*
Lists a file. If the file is a directory lists files in the directory.

mail
Read and send mail

man *command*
View manual pages for a command.

mkdir *directory*
Creates a new directory.

mount -F *type* /dev/dsk/*device* /*mountpoint*
Mounts the filesystem of type from /dev/device to /mnt/mounpoint

mount -F hsfs -o ro /dev/dsk/c0t0d0s0 / cdrom
Mount the CD-ROM drive so that the files can be read as part of the normal file structure. The device name may be different depending upon the disk setups.

mv *file1 file2*
Moves or renames a file or directory.

passwd
Changes the password

pg *file1*
View a text file one page at a time

ps -ef
Show all processes

pwd
Shows the current working directory.

rm *file1*
Deletes (unlinks) a file

rm -r *file1*
Removes a directory (including all files and subdirectories)

rmdir *directory*
Removes a directory and it’s contents

sed *file*
    Edits a file using the stream editor.

shutdown *time*
    Shutdown the computer at the specified time interval

stty
    Sets terminal settings

stty sane
    Resets terminal to default settings

tail -n *count file*
    Shows count number of lines from the bottom of file
tail -f *file*
    Shows bottom of file, showing new lines as they are added.

tar -c *file1 file2*
    Archives file1 an file2 to the default backup device (normally a tape drive)

tar -cvf *filename.tar file1 file2*
    Archive file1 and file2 into a file called filename.tar. The -v option will list all files archived

tar -x
    Extract the files from the default backup device.

tar -xvf *filename.tar*
    Extract all files from an archive. The -v option shows all the files as they are extracted.

telnet *hostname*
    Logs into a remote system

touch *file*
    Updates the access time for a file. If the file does not exist then it creates an empty file.

umount *directory*
    Unmounts the directory

umount -f *device*
    Unmounts using the device name. The -f option forces the umount if the filesystem is in use.

uname
    Shows the name and version of the operating system.

ucompress *filename*

Uncompresses a file compressed using the compress file. Files will normally be suffixed with .Z prior to being uncompressed.

vi *file*
Edits a file using the vi editor

who
Displays users on a system

who am i
Displays your username

Index

$ 19
$HOME 59
$PATH 60
$TERM 59
% 19
& 75
. 35
.. 35
.kshrc 64
.profile **63**
/ 33
| 53, 75
< 56, 75
> 54, 75
>> 54
absolute directory 35
access 75
account 57
ACL 75
ACLST 75
AIX 5, 75
alias 75
ANSI 75
application 75
apropos **27**
argument 23
ASCII 75
at 81
authorised 9
awk 75
background process 75
Backspace 63
BACKSPACE 14
backup 44, 75
bash 18
Bash **18**
bin **34**
binary 48
block device 75
booting 75
Bourne **18**
BSD 4, 75
byte 75
C 75
C Shell **18**
case sensitive 9, **14**
cat **45**, 81
catman 26, 27
cd 23, **35**, 36, 81
CDE 5, 10, 27, 75
CGI 75
character 76
chgrp 81
child 76
chmod 47, 48, 81
chown 49, 81
CISC 76
classic file 33
Clear 29
client 76
command **21**, 23, 46, **51**, 76
command line 23
Command Window 10
commands **28**
Common Desktop Environment *See* CDE
compress 81
console 76
copy 45
cp 23, **44**, 81
cpio 45
create 41
csh 18
CUT 76
data 31
date 43
del 82
delete 44
DELETE 14
deleted 44
dev **34**
device driver 76
devices **34**
df 82
diff 82
directory 28, **31**, 33, 41, 46, 76
disk 33, 76
diskette 76
disks 31
DoS 76
dthelpview 27
du 82
e 82
EBCDIC 76

Echo 29
ed82
edit 67
editor 76
env 82
ENV 62, 64
environment 76
escape 76
etc **34**
Ethernet 76
execute 47
execute permission 76
Exiting 13
export **34**, 64
field seperator 76
FIFO 76
file **31**, 37, 41, **46**, 76
file system 76
filename 33
find **28**, 82
flag 76
fork 76
ftp 82
gateway 77
GMT 77
grep 82
group 38, 46, 77
gunzip 82
gzip 82
hardware 77
Hardware 3
head **45**, 82
help 24, 27, 82
heterogeneous 77
Hewlett-Packard 5
hidden file **38**
history 65
home **33**, 35
homogeneous 77
hostname 12
HP-UX 5
IBM 5
ICONS 10
interpreter 77
IP 77
IP address 12
IPL 77
Journaled File System 77
kernel 17, **34**, 77
keyboard 14, 54
kill 77, 82
Korn **18**
ksh 18, 64
LAN 77
lib **34**
line editor 77
link 77
Linux 5, 77
ln 83
logfile 45
login 12, 77
Login **9**
LOGNAME 62
logout 13, 77
Lost + Found **38**
lp 46
ls 23, 24, **36**, 83. *See*
mail 83
MAIL 62
MAILCHECK 62
make 77
man 24, **25**, 83
Manual Pages *See* man
memory 77
metacharacters 43, 77
mkdir **43**, 83
mnt **34**
modify 41
more 45
Motif 77
mount 83
move 44
multitasking 4
mv **44**, 83
network 11
NFS 34, 77
noclobber 56
null 77
octal 48
OEM 78
operating system 33, 78
Operating System **1**
opt **34**
option 23
options 23, 53
OS/2 11
others 46
overwrite 56
overwriting 45
owner 49, 78

parallel processing......................... 78
passwd 83
password..............................9, 10, 78
PATH.......................................62, 78
path name 76, 78
PC... 11
permission......................38, 46, 75, 78
pg.................................23, 24, 45, 83
pipe....................................23, 43, 53
pipes.. 78
POSIX... 78
print **46**
printer 54
proc.. **34**
process 78
profile.. 78
program **21**
programming.................................. 18
prompt 10, 13, **19**
ps 83
PS1 62
pwd..**36**, 83
read 47
read permission 78
redirect... **54**
redirection 78
regular expression......................... 78
relative directory 35
relative path name......................... 78
remove 44
rename 44
replace 72
rexec 11
RISC 78
rlogin 11
rm **43**, **44**, 59, 83
rmdir **43**, 83
root 19, **59**
root directory 33, 78
RPM... 78
rsh.. 11
sbin **34**
scalability...................................... 78
SCCS... 78
SCO 5
script..53, 79
search... 72
Security.. 9
sed 84
server 78
set...64
set -o vi 64
setuid 78
sh 18
shell...................**15**, **18**, 19, 43, 59, 78
shutdown 84
SMIT...5
sockets 79
software 79
Solaris 5
Solaris Manuals 28
SPARC 5
special file **33**
stderr **54**
STDERR 79
stdin...**54**
STDIN 79
stdout...**54**
STDOUT 79
stored..31
stty..................................... 14, 63, 84
su **59**
subdirectory...............................34, 79
Sun 5
SunOS 5
superuser 57, 79
swap space......................................79
switches...53
SwitchUser ....*See* su
system 79
System V...79
tail **45**, 84
tar...84
TCP 79
TCP/IP.......................................11, 79
tcsh..**18**
tee..**55**
telnet.................................. 11, 60, 84
TERM..62
termcap 79
text 33, 46, 67
thread 4
tmp **34**
touch...**43**, 84
type 46
ucompress..84
umask 62
umount 84
uname 84
undelete .... *See*

UNIX....................................**1**, 17, 79
user 46
username...............................9, 35, 59
usr.. **33**
var.. **34**
variables... 59
vi **67**, 79, 85
view... 45
whatis database 26
who.. 85
wild card....................................... 79
window 79
Windows 11, 60
working directory 79
Workstation.......................................9
write 47
write permission............................80
X *See* X Windows
X Windows...................... 5, 64, 69, 80
x86 5
X-Windows 23